Package 'FastKRR'

September 22, 2025

Type Package

Title Kernel Ridge Regression using 'RcppArmadillo'

Version 0.1.0

Description Provides core computational operations in C++ via 'RcppArmadillo', enabling faster performance than pure R, improved numerical stability, and parallel execution with OpenMP where available. On systems without OpenMP support, the package automatically falls back to single-threaded execution with no user configuration required. For efficient model selection, it integrates with 'CVST' to provide sequential-testing cross-validation that identifies competitive hyperparameters without exhaustive grid search. The package offers a unified interface for exact kernel ridge regression and three scalable approximations—Nyström, Pivoted Cholesky, and Random Fourier Features—allowing analyses with substantially larger sample sizes than are feasible with exact KRR. It also integrates with the 'tidymodels' ecosystem via the 'parsnip' model specification 'krr_reg', the S3 method tunable.krr_reg(), and the direct fitting helper fit_krr(). To understand the theoretical background, one can refer to Wainwright (2019) <doi:10.1017/9781108627771>.

License GPL (>= 2)

URL https://github.com/kybak90/FastKRR, https://www.tidymodels.org

BugReports https://github.com/kybak90/FastKRR/issues

Imports CVST, generics, parsnip, Rcpp, rlang, tibble

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, dials, tidymodels, modeldata, dplyr

SystemRequirements OpenMP (optional)

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Author Gyeongmin Kim [aut] (Sungshin Women's University),
Seyoung Lee [aut] (Sungshin Women's University),
Miyoung Jang [aut] (Sungshin Women's University),
Kwan-Young Bak [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-4541-160X>, Sungshin Women's
University)

Maintainer Kwan-Young Bak <kybak@sungshin.ac.kr>

Repository CRAN

Date/Publication 2025-09-22 11:20:12 UTC

Contents

`FastKRR-package` *Kernel Ridge Regression using the* **RcppArmadillo** *Package*

Description

The **FastKRR** implements its core computational operations in C++ via **RcppArmadillo**, enabling faster performance than pure R, improved numerical stability, and parallel execution with OpenMP where available. On systems without OpenMP support, the package automatically falls back to single-threaded execution with no user configuration required. For efficient model selection, it integrates with **CVST** to provide sequential-testing cross-validation that identifies competitive hyperparameters without exhaustive grid search. The package offers a unified interface for exact kernel ridge regression and three widely used scalable approximations—Nyström, Pivoted Cholesky, and Random Fourier Features—allowing analyses with substantially larger sample sizes than are feasible with exact KRR while retaining strong predictive performance. This combination of a compiled backend and scalable algorithms addresses limitations of packages that rely solely on exact computation, which is often impractical for large n. It also integrates with the **tidymodels** ecosystem via the **parsnip** model specification `krr_reg`, the S3 method `tunable.krr_reg()` (exposes tunable parameters to `dials`/`tune`), and the direct fitting helper `fit_krr()`; see their help pages for usage.

Directory structure

- `R/`: High-level R functions and user-facing API
- `src/`: C++ sources (kernel computation, fitting, prediction)

This package links against **Rcpp** and **RcppArmadillo** (via LinkingTo). It uses **CVST**, **parsnip**, and the **tidymodels** ecosystem through their public R APIs.

Author(s)

Maintainer: Kwan-Young Bak <kybak@sungshin.ac.kr> (ORCID) (Sungshin Women's University) [copyright holder]

Authors:

- Gyeongmin Kim <rlarudals0824@gmail.com> (Sungshin Women's University)
- Seyoung Lee <sudang0404@gmail.com> (Sungshin Women's University)
- Miyoung Jang <miyoung9072@gmail.com> (Sungshin Women's University)

See Also

CVST, **Rcpp**, **RcppArmadillo**, **parsnip**, **tidymodels**

`approx_kernel` *Compute low-rank approximations(Nyström, Pivoted Cholesky, RFF)*

Description

This function compute low-rank kernel approximation $\tilde{K} \in \mathbb{R}^{n \times n}$using three methods: Nyström approximation, Pivoted Cholesky decomposition, and Random Fourier Features (RFF).

Usage

```
approx_kernel(
  K = NULL,
  X = NULL,
  opt = c("nystrom", "pivoted", "rff"),
  kernel = c("gaussian", "laplace"),
  m = NULL,
  d,
  rho,
  eps = 1e-06,
  W = NULL,
  b = NULL,
  n_threads = 4
)
```

Arguments

`K`	Exact Kernel matrix $K \in \mathbb{R}^{n \times n}$. Used in `"nystrom"` and `"pivoted"`.
`X`	Design matrix $X \in \mathbb{R}^{n \times d}$. Only required for `"rff"`.
`opt`	Method for constructing or approximating :
	`"nystrom"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using the Nyström approximation.

"pivoted" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using Pivoted Cholesky decomposition.

"rff" Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using Random Fourier Features (RFF).

`kernel` Kernel type either "gaussian"or "laplace".

`m` Approximation rank (number of random features) for the low-rank kernel approximation. If not specified, the recommended choice is

$$\lceil n \cdot \log(d+5)/10 \rceil$$

where X is design matrix, $n = nrow(X)$ and $d = ncol(X)$.

`d` Design matrix's dimension ($d = ncol(X)$).

`rho` Scaling parameter of the kernel (ρ), specified by the user.

`eps` Tolerance parameter used only in "pivoted" for stopping criterion of the Pivoted Cholesky decomposition.

`W` Random frequency matrix $\omega \in \mathbb{R}^{m \times d}$

`b` Random phase vector $b \in \mathbb{R}^m$, i.i.d. $\mathrm{Unif}[0,\ 2\pi]$.

`n_threads` Number of parallel threads. The default is 4. If the system does not support 4 threads, it automatically falls back to 1 thread. It is applied only for `opt = "nystrom"` or `opt = "rff"` , and for the Laplace kernel (`kernel = "laplace"`).

Details

Requirements and what to supply:

Common • `d` and `rho` must be provided (non-`NULL`).

`nystrom` / `pivoted` • Require a precomputed kernel matrix `K`; error if K is `NULL`.
- If `m` is `NULL`, use $\lceil n \cdot \log(d+5)/10 \rceil$.
- For "pivoted", a tolerance `eps` is used; the decomposition stops early when the next pivot (residual diagonal) drops below `eps`.

`rff` • K must be `NULL` (not used) and `X` must be provided with `d = ncol(X)`.
- To pre-supply random features, provide both `W` (random frequency matrix $\omega \in \mathbb{R}^{m \times d}$) and `b` (random phase vector $b \in \mathbb{R}^m$); error if only one is supplied or shapes mismatch.
- When `W` and `b` are supplied by `rff_random`.

Value

A list containing the results of the low-rank kernel approximation. The exact structure depends on the chosen method:

"nystrom" • `K_approx`: Approximated kernel matrix ($K \in \mathbb{R}^{n \times n}$) from the Nyström approximation.
- `m`: Approximation rank used for the low-rank kernel approximation.
- `n_threads`: Number of threads used in the computation.
- `method`: The kernel approximation method actually used. The string "nystrom".

`"pivoted"` • `K_approx`: Approximated kernel matrix ($K \in \mathbb{R}^{n \times n}$) from the Pivoted Cholesky decomposition.

- `m`: Effective rank actually used. This value is at most the requested `m` and may be smaller if early stopping is triggered by `eps`
- `method`: The kernel approximation method actually used. The string `"pivoted"`.

`"rff"` • `K_approx`: Approximated kernel matrix ($K \in \mathbb{R}^{n \times n}$).

- `method`: The kernel approximation method actually used. The string `"rff"`.
- `m`: Number of random features used.
- `d`: Input design matrix's dimension.
- `rho`: Scaling parameter of the kernel.
- `W`: Random frequency matrix ($m \times d$).
- `b`: Random phase vector (m).
- `used_supplied_Wb`: Logical; `TRUE` if user-supplied `W`, `b` were used, `FALSE` otherwise.
- `n_threads`: Number of threads used in the computation.

Examples

```
# data setting
set.seed(1)
d = 1
n = 1000
m = 50
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = as.vector(sin(2*pi*rowMeans(X)^3) + rnorm(n, 0, 0.1))
K = make_kernel(X, kernel = "gaussian", rho = 1)

# Example: RFF approximation
rff_pars = rff_random(m = m, d = d, rho = 1, kernel = "gaussian")
K_rff = approx_kernel(X = X, opt = "rff", kernel = "gaussian",
                      m = m, d = d, rho = 1,
                      W = rff_pars$W, b = rff_pars$b,
                      n_threads = 1)

# Exapmle: Nystrom approximation
K_nystrom = approx_kernel(K = K, opt = "nystrom",
                          m = m, d = d, rho = 1,
                          n_threads = 1)

# Example: Pivoted Cholesky approximation
K_pivoted = approx_kernel(K = K, opt = "pivoted",
                          m = m, d = d, rho = 1)
```

`fastkrr` *Fit kernel ridge regression using exact or approximate methods*

Description

This function performs kernel ridge regression (KRR) in high-dimensional settings. The regularization parameter λ can be selected via the CVST (Cross-Validation via Sequential Testing) procedure. For scalability, three different kernel approximation strategies are supported (Nyström approximation, Pivoted Cholesky decomposition, Random Fourier Features(RFF)), and kernel matrix can be computed using two methods(Gaussian kernel, Laplace kerenl).

Usage

```
fastkrr(
  x,
  y,
  kernel = "gaussian",
  opt = "exact",
  m = NULL,
  eps = 1e-06,
  rho = 1,
  lambda = NULL,
  fastcv = FALSE,
  n_threads = 4,
  verbose = TRUE
)
```

Arguments

`x`	Design matrix $X \in \mathbb{R}^{n \times d}$.
`y`	Response variable $y \in \mathbb{R}^n$.
`kernel`	Kernel type either "gaussian"or "laplace".
`opt`	Method for constructing or approximating :
	`"exact"` Construct the full kernel matrix $K \in \mathbb{R}^{n \times n}$ using design martix X.
	`"nystrom"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using the Nyström approximation.
	`"pivoted"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using Pivoted Cholesky decomposition.
	`"rff"` Use Random Fourier Features to construct a feature map $Z \in \mathbb{R}^{n \times m}$ (with m random features) so that $K \approx ZZ^\top$. Here, m is the number of features.
`m`	Approximation rank(number of random features) used for the low-rank kernel approximation. If not provided by the user, it defaults to $$\lceil n \cdot \frac{\log(d+5)}{10} \rceil,$$ where $n = nrow(X)$ and $d = ncol(X)$.
`eps`	Tolerance parameter used only in `"pivoted"` for stopping criterion of the Pivoted Cholesky decomposition.

`rho` Scaling parameter of the kernel(ρ), specified by the user. Defaults to `1`.

$$\text{Gaussian kernel : } \mathcal{K}(x, x') = \exp(-\rho\|x - x'\|_2^2)$$

$$\text{Laplace kernel : } \mathcal{K}(x, x') = \exp(-\rho\|x - x'\|_1)$$

`lambda` Regularization parameter. If `NULL`, the penalty parameter is chosen automatically via **CVST** package. If not provided, the argument is set to a kernel-specific grid of 100 values: $[10^{-10}, 10^{-3}]$ for Gaussian, $[10^{-5}, 10^{-2}]$ for Laplace.

`fastcv` If `TRUE`, accelerated cross-validation is performed via sequential testing (early stopping) as implemented in the **CVST** package. The default is `FALSE`.

`n_threads` Number of parallel threads. The default is 4. If the system does not support 4 threads, it automatically falls back to 1 thread. Parallelization (implemented in C++) is one of the main advantages of this package and is applied only for `opt = "nystrom"` or `opt = "rff"`, and for the Laplace kernel (`kernel = "laplace"`).

`verbose` If TRUE, detailed progress and cross-validation results are printed to the console. If FALSE, suppresses intermediate output and only returns the final result.

Details

The function performs several input checks and automatic adjustments:

- If `x` is a vector, it is converted to a one column matrix. Otherwise, `x` must be a matrix; otherwise an error is thrown.
- `y` must be a vector, and its length must match `nrow(x)`.
- `kernel` must be either `gaussian` or `laplace`.
- `opt` must be one of `"exact"`, `"pivoted"`, `"nystrom"`, or `"rff"`.
- If `m` is `NULL`, it defaults to
$$\lceil n \cdot \log(d + 5)/10 \rceil$$
where $n = nrow(X)$ and $d = ncol(X)$. Otherwise, `m` must be a positive integer.
- `rho` must be a positive real number (default is 1).
- `lambda` can be specified in three ways:
  1. A positive numeric scalar, in which case the model is fitted with this single value.
  2. A numeric vector (length >= 3) of positive values used as a tuning grid; selection is performed by **CVST** cross-validation (sequential testing if `fastcv = TRUE`).
  3. `NULL`: use a default grid (internal setting) and tune `lambda` via **CVST** cross-validation (sequential testing if `fastcv = TRUE`).
- `n_threads`: Number of threads for parallel computation. Default is `4`. If the system has <= 3 available processors, it uses `1`.

Value

An object of class `fastkrr`. The returned object is a list with components:

`coefficients` Estimated coefficient vector $\mathbb{R}^n$. Accessible via `model$coefficients`.

`fitted.values` Fitted values $\mathbb{R}^n$. Accessible via `model$fitted.values`.

`opt` Kernel approximation option. One of `"exact"`, `"pivoted"`, `"nystrom"`, `"rff"`.

`kernel` Kernel used (`"gaussian"` or `"laplace"`).

`x` Input design matrix.

`y` Response vector.

`lambda` Regularization parameter. If `NULL`, tuned by cross-validation via **CVST**.

`rho` Additional user-specified hyperparameter.

`n_threads` Number of threads used for parallelization.

Additional components depend on the value of `opt`:

opt = "exact":

`K` The full kernel matrix.

opt = "pivoted":

`m` Effective rank actually used. This value is at most the requested `m` and may be smaller if early stopping is triggered by `eps`.

`K` Exact kernel matrix $K \in \mathbb{R}^{n\times n}$ computed by `make_kernel(..., opt = "exact")`.

`PR` The method provides a low-rank approximation to the kernel matrix $PR \in \mathbb{R}^{n\times m}$ obtained via Pivoted Cholesky decomposition; satisfies $K \approx PR\,(PR)^\top$.

opt = "nystrom":

`m` Approximation rank used for the low-rank kernel approximation.

`K` Exact kernel matrix $K \in \mathbb{R}^{n\times n}$ computed by `make_kernel(..., opt = "exact")`.

`R` The method provides a low-rank approximation to the kernel matrix $R \in \mathbb{R}^{n\times m}$ obtained via Nyström approximation; satisfies $K \approx RR^\top$.

opt = "rff":

`m` Number of random features.

`z` Random Fourier Feature matrix $Z \in \mathbb{R}^{n\times m}$ with $Z_{ij} = z_j(x_i) = \sqrt{2/m}\cos(\omega_j^\top x_i + b_j), \quad j = 1, \cdots, m$, so that $K \approx ZZ^\top$.

`w` Random frequency matrix $\omega \in \mathbb{R}^{m\times d}$ (row j is $\omega_j^\top \in \mathbb{R}^d$), drawn i.i.d. from the spectral density of the chosen kernel:

- Gaussian: $\omega_{jk} \sim \mathcal{N}(0, 2\gamma)$ (e.g., $\gamma = 1/\ell^2$).
- Laplace: $\omega_{jk} \sim \mathrm{Cauchy}(0, 1/\sigma)$ i.i.d.

`b` Random phase vector $b \in \mathbb{R}^m$, i.i.d. $\mathrm{Unif}[0,\, 2\pi]$.

Examples

```
set.seed(1)
lambda = 1e-4
d = 1
rho = 1
n = 50
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = as.vector(sin(2*pi*rowMeans(X)^3) + rnorm(n, 0, 0.1))
```

```
# Exapmle: pivoted cholesky
model = fastkrr(X, y, kernel = "gaussian", opt = "pivoted", rho = rho, lambda = 1e-4)

# Example: nystrom
model = fastkrr(X, y, kernel = "gaussian", opt = "nystrom", rho = rho, lambda = 1e-4)

# Example: random fourier features
model = fastkrr(X, y, kernel = "gaussian", opt = "rff", rho = rho, lambda = 1e-4)

# Example: Laplace kernel
model = fastkrr(X, y, kernel = "laplace", opt = "nystrom", n_threads = 1, rho = rho)
```

`fit_krr` *Fit Kernel Ridge Regression*

Description

'fit_krr()' fits Kernel Ridge Regression (KRR) either from a design matrix (`x`) with a response (`y`), or via a formula interface. The function is designed to be used with the tidymodels stack. In particular, this package provides a **parsnip** model specification `krr_reg()` with the engine `"fastkrr"`, so you can fit KRR inside **recipes/workflows** pipelines just like any other tidymodels model.

Usage

```
fit_krr(x, ...)

## Default S3 method:
fit_krr(x, y, ...)

## S3 method for class 'formula'
fit_krr(x, data, intercept = FALSE, ...)
```

Arguments

`x`	For the base method, a numeric design matrix $X \in \mathbb{R}^{n \times d}$. For the formula method, a model formula.
`...`	Additional arguments passed to the underlying engine (e.g., `"kernel"`, `"rho"`, `"penalty"`, `"opt"`, `"m"`, `"fastcv"`, etc.).
`y`	Response vector $y \in \mathbb{R}^n$.
`data`	A data frame (formula method).
`intercept`	If `FALSE`, the formula method removes the intercept term by updating the terms to `.~.-1`. Default: `FALSE`.

Value

A fitted KRR object (class includes `"krr"`).

Tidymodels integration

- Use `krr_reg(mode = "regression", ...) %>% parsnip::set_engine("fastkrr")` to select the FastKRR engine.
- Combine with **recipes** and **workflows** to build modular pipelines for preprocessing, resampling, and evaluation.

See Also

`krr_reg`, **parsnip**, **workflows**, **recipes**, **tidymodels**

Examples

```
if (all(vapply(
  c("parsnip","stats","modeldata"),
  requireNamespace, quietly = TRUE, FUN.VALUE = logical(1)
))) {
library(modeldata)
library(dplyr)

# Example : matrix interface
# Data analysis
data(ames)
ames = ames %>% mutate(Sale_Price = log10(Sale_Price))

set.seed(502)
x = as.matrix(ames[, c("Longitude", "Latitude")])
y = ames[, "Sale_Price", drop = TRUE]

fit1 = fit_krr(
  x, y,
  kernel = "gaussian",
  opt    = "exact",
  rho    = 1,
  lambda = 1e-4
)

# Example : formula interface
fit2 = fit_krr(
  Sale_Price ~ .,
  data   = ames,
  kernel = "gaussian",
  opt    = "exact",
  rho    = 1,
  lambda = 1e-4
)
}
```

krr_reg *Kernel Ridge Regression*

Description

‘krr_reg()‘ defines a Kernel Ridge Regression (KRR) model specification for use with the tidymodels ecosystem via **parsnip**. This spec can be paired with the `"fastkrr"` engine implemented in this package to fit exact or kernel approximation (Nyström, Pivoted Cholesky, Random Fourier Features) within **recipes/workflows** pipelines.

Usage

```
krr_reg(
  mode = "regression",
  kernel = NULL,
  opt = NULL,
  eps = NULL,
  n_threads = NULL,
  m = NULL,
  rho = NULL,
  penalty = NULL,
  fastcv = NULL
)
```

Arguments

`mode`	A single string; only ‘"regression"‘ is supported.
`kernel`	Kernel matrix K has two kinds of Kernel ("gaussian", "laplace").
`opt`	Method for constructing or approximating :
	`"exact"` Construct the full kernel matrix $K \in \mathbb{R}^{n \times n}$ using design matrix X.
	`"nystrom"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using the Nyström approximation.
	`"pivoted"` Construct a low-rank approximation of the kernel matrix $K \in \mathbb{R}^{n \times n}$ using Pivoted Cholesky decomposition.
	`"rff"` Use Random Fourier Features to construct a feature map $Z \in \mathbb{R}^{n \times m}$ (with m random features) so that $K \approx ZZ^\top$. Here, m is the number of features.
`eps`	Tolerance parameter used only in `"pivoted"` for stopping criterion of the Pivoted Cholesky decomposition.
`n_threads`	Number of parallel threads. It is applied only for `opt = "nystrom"` or `opt = "rff"`, and for the Laplace kernel (`kernel = "laplace"`).
`m`	Approximation rank(number of random features) used for the low-rank kernel approximation.
`rho`	Scaling parameter of the kernel(ρ).

penalty	Regularization parameter.
fastcv	If TRUE, accelerated cross-validation is performed via sequential testing (early stopping) as implemented in the **CVST** package.

Value

A parsnip model specification of class "krr_reg".

Examples

```
if (all(vapply(
  c("parsnip","stats","modeldata"),
  requireNamespace, quietly = TRUE, FUN.VALUE = logical(1)
))) {
library(tidymodels)
library(parsnip)
library(stats)
library(modeldata)

# Data analysis
data(ames)
ames = ames %>% mutate(Sale_Price = log10(Sale_Price))

set.seed(502)
ames_split = initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train = training(ames_split) # dim (2342, 74)
ames_test  = testing(ames_split) # dim (588, 74)

# Model spec
krr_spec = krr_reg(kernel = "gaussian", opt = "exact",
                   m = 50, eps = 1e-6, n_threads = 4,
                   rho = 1, penalty = tune()) %>%
 set_engine("fastkrr") %>%
 set_mode("regression")

# Define rec
rec = recipe(Sale_Price ~ Longitude + Latitude, data = ames_train)

# workflow
wf = workflow() %>%
  add_recipe(rec) %>%
  add_model(krr_spec)

# Define hyper-parameter grid
param_grid = grid_regular(
  dials::penalty(range = c(-10, -3)),
  levels = 5
)

# CV setting
set.seed(123)
cv_folds = vfold_cv(ames_train, v = 5, strata = Sale_Price)
```

```
# Tuning
tune_results = tune_grid(
  wf,
  resamples = cv_folds,
  grid = param_grid,
  metrics = metric_set(rmse),
  control = control_grid(verbose = TRUE, save_pred = TRUE)
)

# Result check
collect_metrics(tune_results)

# Select best parameter
best_params = select_best(tune_results, metric = "rmse")

# Finalized model spec using best parameter
final_spec = finalize_model(krr_spec, best_params)
final_wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(final_spec)

# Finalized fitting using best parameter
final_fit = final_wf %>% fit(data = ames_train)

# Prediction
predict(final_fit, new_data = ames_test)
print(best_params)

}
```

`make_kernel` *Kernel matrix K construction for given datasets*

Description

Constructs a kernel matrix $K \in \mathbb{R}^{n \times n'}$ given two datasets $X \in \mathbb{R}^{n \times d}$ and $X' \in \mathbb{R}^{n' \times d}$, where $x_i \in \mathbb{R}^d$ and $x'_j \in \mathbb{R}^d$ denote the i-th and j-th rows of X and X', respectively, and $K_{ij} = \mathcal{K}(x_i, x'_j)$ for a user-specified kernel. Implemented in C++ via RcppArmadillo.

Arguments

`X`	Design matrix $X \in \mathbb{R}^{n \times d}$ (rows $x_i \in \mathbb{R}^d$).
`X_new`	Second matrix $X' \in \mathbb{R}^{n' \times d}$ (rows $x'_j \in \mathbb{R}^d$). If omitted, $X' = X$ and $n' = n$.
`kernel`	Kernel type; one of `"gaussian"` or `"laplace"`.
`rho`	Kernel width parameter ($\rho > 0$).

n_threads — Number of parallel threads. The default is 4. If the system does not support 4 threads, it automatically falls back to 1 thread. Parallelization (implemented in C++) is one of the main advantages of this package and is applied only for "laplace" kernels.

Details

Gaussian:

$$\mathcal{K}(x_i, x_j) = \exp\big(-\rho\|x_i - x_j\|_2^2\big)$$

Laplace:

$$\mathcal{K}(x_i, x_j) = \exp\big(-\rho\|x_i - x_j\|_1\big)$$

Value

If X_new is NULL, a symmetric matrix $K_{ij} = \mathcal{K}(x_i, x_j), K \in \mathbb{R}^{n\times n}$. Otherwise, a matrix $K'_{ij} = \mathcal{K}(x_i, x'_j), K' \in \mathbb{R}^{n'\times n}$.

Examples

```
set.seed(1)
lambda = 1e-4
d = 1
rho = 1
n = 1000
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)

# second matrix
new_n = 1500
new_X = matrix(runif(new_n*d, 0, 1), nrow = new_n, ncol = d)

# make kernel : Gaussian kernel
K = make_kernel(X, kernel = "gaussian", rho = rho)
new_K = make_kernel(X, new_X, kernel = "gaussian", rho = rho)

# make kernel : Laplace kernel
K = make_kernel(X, kernel = "laplace", rho = rho, n_threads = 1)
new_K = make_kernel(X, new_X, kernel = "laplace", rho = rho, n_threads = 1)
```

---

make_Z — *Random Fourier Feature matrix Z construction for given datasets*

Description

Constructs a Random Fourier Feature matrix $Z \in \mathbb{R}^{n\times m}$ with $Z_{ij} = z_j(x_i) = \sqrt{2/m}\cos(\omega_j^\top x_i + b_j), \quad j = 1, \cdots, m, \quad i = 1, \cdots, n$. Implemented in C++ via RcppArmadillo.

Arguments

X	Design matrix $X \in \mathbb{R}^{n \times d}$ (rows $x_i \in \mathbb{R}^d$).
W	Random frequency matrix $\omega \in \mathbb{R}^{m \times d}$ (row j is $\omega_j^\top \in \mathbb{R}^d$), drawn i.i.d. from the spectral density of the chosen kernel: • Gaussian: $\omega_{jk} \sim$ i.i.d. $\mathcal{N}(0, 2\rho)$. • Laplace: $\omega_{jk} \sim$ i.i.d. $\mathrm{Cauchy}(0, 1/2\rho)$
b	Random phase vector $b \in \mathbb{R}^m$, i.i.d. $\mathrm{Unif}[0,\ 2\pi]$.
n_threads	Number of parallel threads. The default is 4. If the system does not support 4 threads, it automatically falls back to 1 thread.

Value

Random Fourier Feature matrix Z

Examples

```
set.seed(1)
lambda = 1e-4
d = 1
rho = 1
n = 50
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
m = ceiling(n* log(d + 5)/ 10)
y = as.vector(sin(2*pi*rowMeans(X)^3) + rnorm(n, 0, 0.1))
rv = rff_random(m = m, d = d, rho = 1, kernel = "gaussian")
str(rv)

Z = make_Z(X, rv$W, rv$b)
str(Z)
```

---

pred_krr *Predict responses for new data using fitted KRR model*

Description

'pred_krr()' generates predictions from a fitted Kernel Ridge Regression (KRR) model for new data.

Usage

```
pred_krr(model, newdata)
```

Arguments

model	A fitted KRR model object returned by fastkrr.
newdata	New design matrix or data frame containing new observations for which predictions are to be made.

Details

The kernel matrix between training data and new data is explicitly computed using make_kernel, and predictions are obtained by multiplying this kernel matrix with the fitted coefficients.

Mathematically, predictions are given by

$$\hat{y}_{new} = K_{new}\hat{\alpha}$$

where K_{new} is the kernel matrix and $\hat{\alpha}$ are the estimated coefficients.

Value

A numeric vector of predicted values corresponding to 'newdata'.

See Also

fastkrr, make_kernel

Examples

```
# Fitting model: pivoted
n = 30
d = 1
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
y = as.vector(sin(2*pi*rowMeans(X)^3) + rnorm(n, 0, 0.1))
lambda = 1e-4
rho = 1
model = fastkrr(X, y, kernel = "gaussian", rho = rho, lambda = lambda, opt = "pivoted")

# Predict
new_n = 50
new_x = matrix(runif(new_n*d, 0, 1), nrow = new_n, ncol = d)
new_y = as.vector(sin(2*pi*rowMeans(new_x)^3) + rnorm(new_n, 0, 0.1))

pred = pred_krr(model, new_x)
crossprod(pred, new_y) / new_n
```

---

rff_random *Generate random Fourier features parameters*

Description

This function generates random frequency matrix ω and offsets random phase vector b used in the Random Fourier Features (RFF) method.

Usage

```
rff_random(m, d = d, rho = 1, kernel = "gaussian")
```

Arguments

m	Number of random features m used in the RFF approximation. If not specified, the recommended choice is $$\left\lceil n \cdot \frac{\log(d+5)}{10} \right\rceil$$ where X is design matrix, $n = nrow(X)$ and $d = ncol(X)$.
d	Design matrix's dimension ($d = ncol(X)$).
rho	Kernel width parameter (ρ), specified by the user. Controls the scale of the kernel function. Defaults to 1.
kernel	Kernel matrix $K \in \mathbb{R}^{n \times n}$ has two kinds of Kernel ("gaussian", "laplace").

Details

Notation. Let $X \in \mathbb{R}^{n \times d}$ denote the design matrix of inputs on which RFF will be applied downstream (with $n = nrow(X)$, $d = ncol(X)$). A commonly recommended choice for the number of random features is

$$\left\lceil n \cdot \frac{\log(d+5)}{10} \right\rceil.$$

Value

A list with components:

w Random frequency matrix $\omega \in \mathbb{R}^{d \times m}$, sampled i.i.d. from a Cauchy distribution.

b Random phase vector $b \in \mathbb{R}^m$, i.i.d. $\mathrm{Unif}[0,\, 2\pi]$.

Examples

```
set.seed(1)
lambda = 1e-4
d = 1
rho = 1
n = 1000
X = matrix(runif(n*d, 0, 1), nrow = n, ncol = d)
m = ceiling(n* log(d + 5)/ 10)
y = as.vector(sin(2*pi*rowMeans(X)^3) + rnorm(n, 0, 0.1))
rv = rff_random(m = m, d = d, rho = 1, kernel = "gaussian")
str(rv)
```

`tunable.krr_reg` *Expose tunable parameters for 'krr_reg'*

Description

Supplies a tibble of tunable arguments for 'krr_reg()'.

Usage

```
## S3 method for class 'krr_reg'
tunable(x, ...)
```

Arguments

`x`	A 'krr_reg' model specification.
`...`	Not used; included for S3 method compatibility.

Value

A tibble (one row per tunable parameter) with columns 'name', 'call_info', 'source', 'component', and 'component_id'.

Index